08

Haga lo que hicimos en aula: consultas avanzadas

En esta aula vamos a realizar consultas utilizando recursos de JPA que nos permiten realizar consultas con funciones de agregación tipo `SUM()`, `AVG()`, `MIN()` ,... antes de proceder debemos agregar unos últimos detalles en la clase pedido y ítem pedido que nos permiten obtener el valor total.

- Primero vamos a asignar el nombre de la columna con la anotación @Column(name="columna").

```
@Column(name="valor_total")
private BigDecimal valorTotal=new BigDecimal(0);
```

COPIA EL CÓDIGO

- Vamos a agregar un nuevo método que nos permite relacionar itemPedido con Pedido, así como Pedido con ItemPedido. Además de las anotaciones tenemos que asignar el valor correcto en nuestro relacionamiento.

```
public void agregarItems(ItemsPedido item) {
    item.setPedido(this);
    this.items.add(item);
this.valorTotal= this.valorTotal.add(item.getValor());
}
```

COPIA EL CÓDIGO

- En la clase itemPedido agregamos un método que nos permite calcular el valor total a partir del precio y de la cantidad.

```
public BigDecimal getValor() {
    return this.precioUnitario.multiply(new BigDecimal(this.ca
}
```

COPIA EL CÓDIGO

- Por último en el DAO vamos agregar nuestros métodos de consulta. Vamos a utilizar las funciones de agregación para consultar el valor total en la base de datos con la función `SUM()` .

```
public BigDecimal valorTotalVendido() {
    String jpql= "SELECT SUM(p.valorTotal) FROM Pedido p";
    return em.createQuery(jpql,BigDecimal.class).getSingleResu
}
```

COPIA EL CÓDIGO

- Y también el valor promedio con la función `AVG()` .

```
public Double valorPromedioVendido() {
    String jpql= "SELECT AVG(p.valorTotal) FROM Pedido p";
    return em.createQuery(jpql,Double.class).getSingleResult()
}
```

COPIA EL CÓDIGO

- Otro tipo de consulta es cuando queremos obtener en nuestra consulta atributos de múltiples entidades en una única consulta e incluso funciones de agregación para eso tenemos dos(2) métodos un crearemos un método que retorna una lista de Objetos para formar el relatorio.

```
public List<Object[]> relatorioDeVentas(){
    String jpql="SELECT producto.nombre, "
            + "SUM(item.cantidad), "
            + "MAX(pedido.fecha) "
            + "FROM Pedido pedido "
            + "JOIN pedido.items item "
            + "JOIN item.producto producto "
            + "GROUP BY producto.nombre "
            + "ORDER BY item.cantidad DESC";
    return em.createQuery(jpql,Object[].class).getResultList()
}
```

COPIA EL CÓDIGO

- Y la forma más recomendada es mediante la construcción de una clase VO (value object) es una clase que nos permite enviar información dentro de nuestra aplicación.

```
public class RelatorioDeVenta {

    private String nombreDelProducto;
    private Long CantidadDeProducto;
    private LocalDate FechaDeUltimaVenta;

    public RelatorioDeVenta(String nombreDelProducto, Long can
        this.nombreDelProducto = nombreDelProducto;
        CantidadDeProducto = cantidadDeProducto;
        FechaDeUltimaVenta = fechaDeUltimaVenta;
    }

    public String getNombreDelProducto() {
        return nombreDelProducto;
```

```
    }

    public void setNombreDelProducto(String nombreDelProducto)
        this.nombreDelProducto = nombreDelProducto;
    }

...

public List<RelatorioDeVenta> relatorioDeVentasVO(){
    String jpql="SELECT new com.latam.alura.tienda.vo.Relatori
            + "SUM(item.cantidad), "
            + "MAX(pedido.fecha)) "
            + "FROM Pedido pedido "
            + "JOIN pedido.items item "
            + "JOIN item.producto producto "
            + "GROUP BY producto.nombre "
            + "ORDER BY item.cantidad DESC";
    return em.createQuery(jpql,RelatorioDeVenta.class).getResu
}
```

COPIA EL CÓDIGO

- En una clase de prueba vamos a conseguir visualizar el resultado de las consultas.

Método 1)

```
BigDecimal valorTotal=pedidoDao.valorTotalVendido();
System.out.println("Valor Total: "+ valorTotal);

List<Object[]> relatorio = pedidoDao.relatorioDeVentasVO();
for(Object[] obj:relatorio){
System.out.println(obj[0]);
```

```
System.out.println(obj[1]);
System.out.println(obj[2]);
}
```

COPIA EL CÓDIGO

Método 2)

```
List<RelatorioDeVenta> relatorio = pedidoDao.relatorioDeVentas\
relatorio.forEach(System.out::println);
```

COPIA EL CÓDIGO

- Para organizar las consultas podemos utilizar un recurso de JPA @NamedQueries que nos permite colocar una determinada consulta dentro de la entidad para indicar que esa consulta tiene un uso particular.

```
@Entity
@Table(name="productos")
@NamedQuery(name="Producto.consultarPrecioPorNombre", query="Sl
public class Producto{...
```

COPIA EL CÓDIGO

- Aun cuando se encuentra indicada en la entidad el método continúa en la clase DAO y la consulta se encuentra en la entidad.

```
public BigDecimal consultarPrecioPorNombreDeProducto(String nor
    return em.createNamedQuery("Producto.consultarPrecioPorNoml
}
```

COPIA EL CÓDIGO